

Exhibit A

COMet Product Information

# How COMet Implements the Model

## Overview

This section describes how COMet implements the interworking model. The following topics are discussed:

- Role of COMet.
- Graphical Overview of Role.
- COM View of CORBA Objects.
- Graphical Overview of View.
- Creating a View.
- Advantages for the COM Programmer.
- Supported Protocols.

## Role of COMet

COMet supports application integration across network boundaries, different operating systems, and different programming languages.

It provides a high performance dynamic bridge that enables integration between COM or Automation and CORBA objects. It allows you to develop and deploy COM or Automation client applications that can interact with existing CORBA server applications that might be running on Windows or another platform.

## Graphical Overview of Role

Figure 3 provides a conceptual overview of how COMet implements the interworking model.

**Figure 3: COMet's Implementation of the Interworking Model**

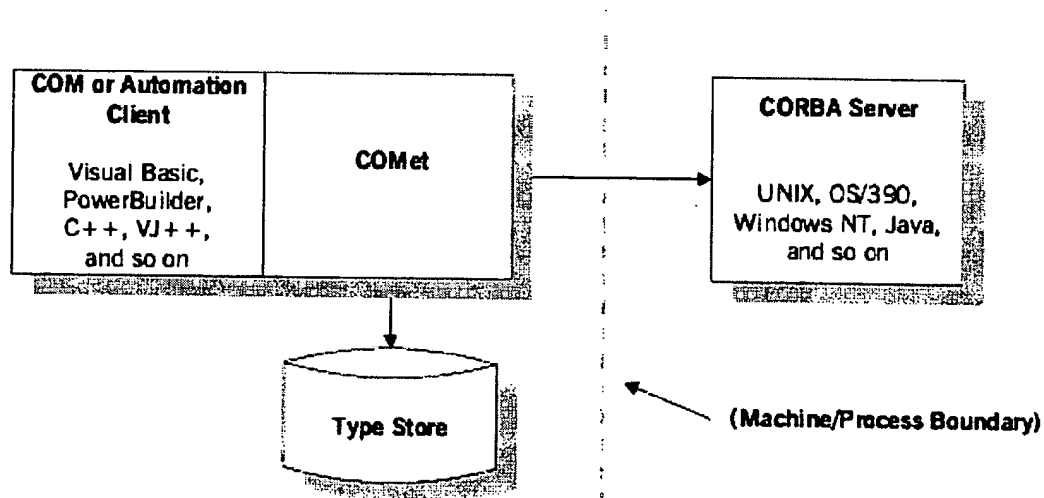


Figure 3 shows no process boundary between the client and COMet, which is the only supported scenario for COM clients. In the case of Automation clients, however, you can choose to have a process and machine boundary between the client and COMet, or to have no machine boundary between COMet and the server. See Usage Models and Bridge Locations for more details.

As explained in Bridge View of Target Object, the interworking

**COM View of  
CORBA Objects**

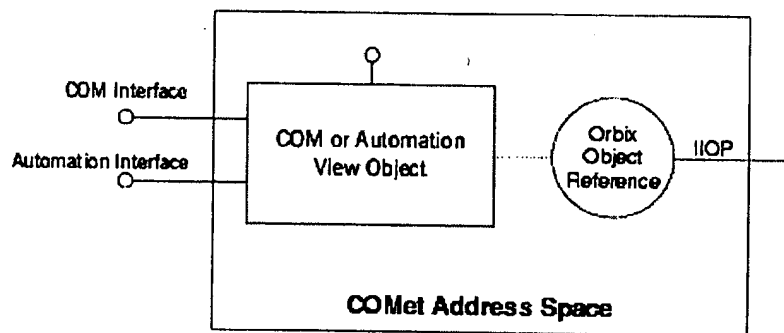
model provides the concept of a view object in the bridge, which allows a client to make requests on an object in a foreign object system as if that object were in the client's own native system. It follows that COMet supports the concept of COM or Automation views of CORBA objects.

This in turn means that a corresponding COM or Automation interface must be generated for each CORBA interface that is implemented by the CORBA objects a client wants to invoke. (COMet supplies utilities that allow you to generate such COM or Automation interfaces from CORBA interfaces, and these are described in more detail in [Development Support Tools](#).) At application runtime, a client can create and subsequently invoke on view objects that implement and expose these COM or Automation interfaces (see [Creating a View](#) for more details).

**Graphical  
Overview of  
View**

[Figure 4](#) provides a graphical overview of how a view object is implemented in COMet.

**Figure 4: View Object in COMet**

**Creating a  
View**

A view object is created in the COMet bridge when a client calls the COMet-supplied (D)ICORBAFactory::GetObject() method on a particular CORBA object. As shown in [Figure 4](#), a view exposes COM or Automation interfaces, which correspond to the CORBA interfaces on the object that the client wants to invoke. The view object is automatically bound on creation to an Orbix object reference for the target object. This object reference is returned to the client, to allow it to invoke operations on the target object. See Part 2 "Programmer's Guide" and [COMet API Reference](#) for more details of how to use D(ICORBA)Factory::GetObject().

**Note:**

All COM views that are mapped from a particular OMG IDL interface must share the same COM IIDs. See [Mapping Interface Identifiers](#) for more details.

**Advantages for  
the COM  
Programmer**

COMet provides two main advantages to COM programmers:

1. COMet provides access to existing CORBA servers, which can be implemented on any operating system and in any language supported by a CORBA implementation. Orbix supports a range of operating systems, such as Windows, UNIX, and OS/390. It also supports different programming languages, including C++ and Java.

2. Using COMet, a COM programmer can use familiar COM-based and Automation-based tools to build heterogeneous systems that use both COM and CORBA components within a COM environment. COMet, therefore, presents a programming model that is familiar to the COM programmer.

**Supported  
Protocols**

COMet supports both the internet inter-ORB protocol (IIOP) and Microsoft's distributed component object model (DCOM) protocol. This means that any IIOP-compliant ORB can interact with a COMet application.

**Note:**

There are some restrictions in the use of DCOM with COMet. These are explained in more detail in [Usage Models and Bridge Locations](#). The recommended approach is to load the bridge in-process to the client (that is, in the client's address space) and hence allow the client machine to use IIOP to communicate with the server.